In a 16x16 grid, select by hand the parameters of a two hidden layer MLP with signum units (threshold nonlinearity) with two inputs and one output to mimic the space partition shown in Fig. 1 (assume that the center of the figure is (0,0)). State the smallest number of hidden units the network needs in each layer and explain their role in creating the mask. Assume that black is -1 and white is 1 (or 0 and 1).

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [174…
```
# with a 16x16 grid,
# we can create the following layers to get
# 1 output node from an mlp with 2 hidden layers
# we know that there need's to be at least 6 neurons to draw the A, as we se
# we can make the second layer represent each stroke of A (there are 3)

# create a 2d array of 0s

output_data = []

def neg_sign(x):
    return 1 if x > 0 else -1

# loop through
for x_1 in np.arange(-8, 9, 1):
    for x_2 in np.arange(-8, 9, 1):
        v_1 = np.sign(8 + (2 * x_1) - x_2)
        v_2 = np.sign(8 - (2 * x_1)- x_2)
        v_3 = np.sign(6 - (2 * x_1) - x_2)
        v_4 = np.sign(6 + (2 * x_1) - x_2)
        v_5 = np.sign(-3 + (0 * x_1) - x_2)
        v_6 = np.sign(-2 + (0 * x_1) - x_2)

        z_1 = np.sign((v_1 - v_4) + v_2/2 - 1)
        z_2 = np.sign((v_2 - v_3) + v_1/2 - 1)
        z_3 = np.sign((v_6 - v_5) +  v_1/2 + v_2/2 - 1)

        output = neg_sign(z_1 + z_2 + z_3 + 2)

        output_data.append([x_1, x_2, output])

        print(f"[{x_1}, {x_2}] -> {output}")


# plot the A_empty, with range -8 to 8
# plot the results
```

```
plt.scatter([x[0] for x in output_data], [x[1] for x in output_data], c=[-x[
```

```
[-8, -8] -> 1
[-8, -7] -> -1
[-8, -6] -> -1
[-8, -5] -> -1
[-8, -4] -> -1
[-8, -3] -> -1
[-8, -2] -> -1
[-8, -1] -> -1
[-8, 0] -> -1
[-8, 1] -> -1
[-8, 2] -> -1
[-8, 3] -> -1
[-8, 4] -> -1
[-8, 5] -> -1
[-8, 6] -> -1
[-8, 7] -> -1
[-8, 8] -> -1
[-7, -8] -> 1
[-7, -7] -> 1
[-7, -6] -> 1
[-7, -5] -> -1
[-7, -4] -> -1
[-7, -3] -> -1
[-7, -2] -> -1
[-7, -1] -> -1
[-7, 0] -> -1
[-7, 1] -> -1
[-7, 2] -> -1
[-7, 3] -> -1
[-7, 4] -> -1
[-7, 5] -> -1
[-7, 6] -> -1
[-7, 7] -> -1
[-7, 8] -> -1
[-6, -8] -> -1
[-6, -7] -> -1
[-6, -6] -> 1
[-6, -5] -> 1
[-6, -4] -> 1
[-6, -3] -> -1
[-6, -2] -> -1
[-6, -1] -> -1
[-6, 0] -> -1
[-6, 1] -> -1
[-6, 2] -> -1
[-6, 3] -> -1
[-6, 4] -> -1
[-6, 5] -> -1
```

```
[-6, 6] -> -1
[-6, 7] -> -1
[-6, 8] -> -1
[-5, -8] -> -1
[-5, -7] -> -1
[-5, -6] -> -1
[-5, -5] -> -1
[-5, -4] -> 1
[-5, -3] -> 1
[-5, -2] -> 1
[-5, -1] -> -1
[-5, 0] -> -1
[-5, 1] -> -1
[-5, 2] -> -1
[-5, 3] -> -1
[-5, 4] -> -1
[-5, 5] -> -1
[-5, 6] -> -1
[-5, 7] -> -1
[-5, 8] -> -1
[-4, -8] -> -1
[-4, -7] -> -1
[-4, -6] -> -1
[-4, -5] -> -1
[-4, -4] -> -1
[-4, -3] -> 1
[-4, -2] -> 1
[-4, -1] -> 1
[-4, 0] -> 1
[-4, 1] -> -1
[-4, 2] -> -1
[-4, 3] -> -1
[-4, 4] -> -1
[-4, 5] -> -1
[-4, 6] -> -1
[-4, 7] -> -1
[-4, 8] -> -1
[-3, -8] -> -1
[-3, -7] -> -1
[-3, -6] -> -1
[-3, -5] -> -1
[-3, -4] -> -1
[-3, -3] -> 1
[-3, -2] -> 1
[-3, -1] -> -1
[-3, 0] -> 1
[-3, 1] -> 1
[-3, 2] -> 1
[-3, 3] -> -1
[-3, 4] -> -1
```

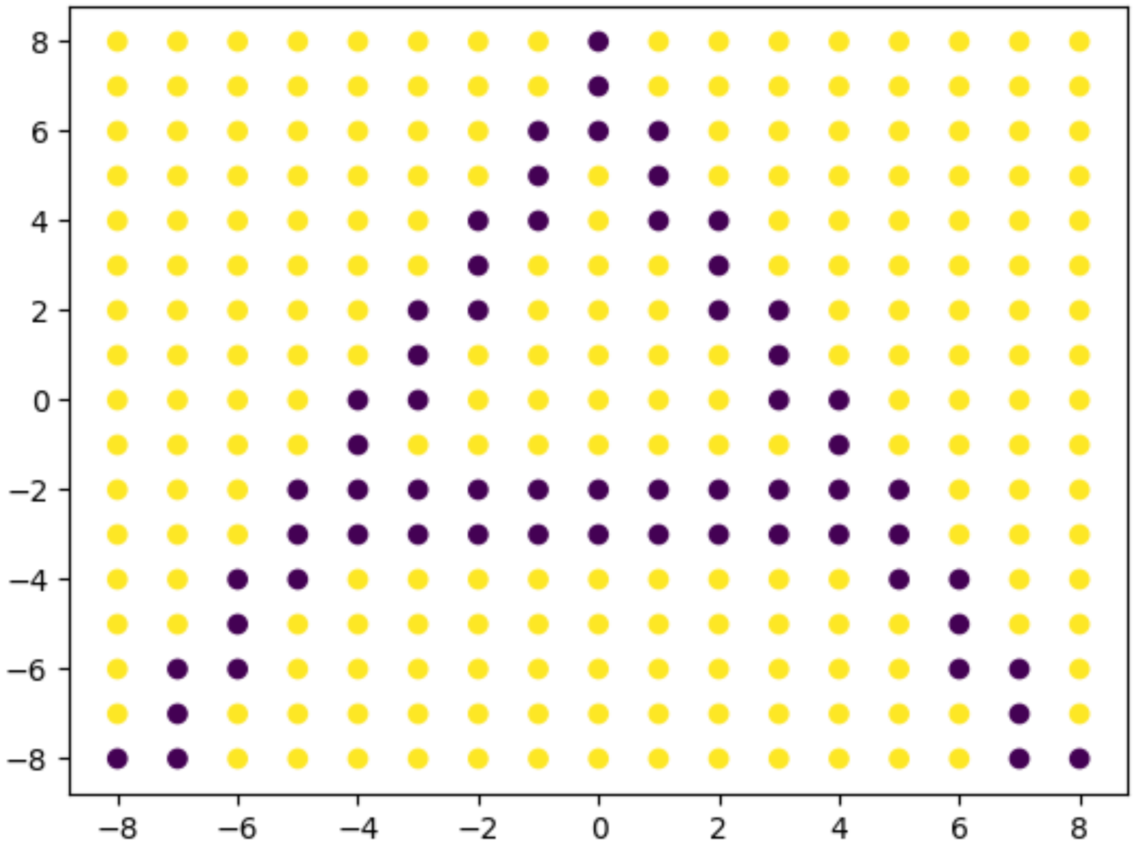
```
[-3, 5] -> -1
[-3, 6] -> -1
[-3, 7] -> -1
[-3, 8] -> -1
[-2, -8] -> -1
[-2, -7] -> -1
[-2, -6] -> -1
[-2, -5] -> -1
[-2, -4] -> -1
[-2, -3] -> 1
[-2, -2] -> 1
[-2, -1] -> -1
[-2, 0] -> -1
[-2, 1] -> -1
[-2, 2] -> 1
[-2, 3] -> 1
[-2, 4] -> 1
[-2, 5] -> -1
[-2, 6] -> -1
[-2, 7] -> -1
[-2, 8] -> -1
[-1, -8] -> -1
[-1, -7] -> -1
[-1, -6] -> -1
[-1, -5] -> -1
[-1, -4] -> -1
[-1, -3] -> 1
[-1, -2] -> 1
[-1, -1] -> -1
[-1, 0] -> -1
[-1, 1] -> -1
[-1, 2] -> -1
[-1, 3] -> -1
[-1, 4] -> 1
[-1, 5] -> 1
[-1, 6] -> 1
[-1, 7] -> -1
[-1, 8] -> -1
[0, -8] -> -1
[0, -7] -> -1
[0, -6] -> -1
[0, -5] -> -1
[0, -4] -> -1
[0, -3] -> 1
[0, -2] -> 1
[0, -1] -> -1
[0, 0] -> -1
[0, 1] -> -1
[0, 2] -> -1
[0, 3] -> -1
```

```
[0, 4] -> -1
[0, 5] -> -1
[0, 6] -> 1
[0, 7] -> 1
[0, 8] -> 1
[1, -8] -> -1
[1, -7] -> -1
[1, -6] -> -1
[1, -5] -> -1
[1, -4] -> -1
[1, -3] -> 1
[1, -2] -> 1
[1, -1] -> -1
[1, 0] -> -1
[1, 1] -> -1
[1, 2] -> -1
[1, 3] -> -1
[1, 4] -> 1
[1, 5] -> 1
[1, 6] -> 1
[1, 7] -> -1
[1, 8] -> -1
[2, -8] -> -1
[2, -7] -> -1
[2, -6] -> -1
[2, -5] -> -1
[2, -4] -> -1
[2, -3] -> 1
[2, -2] -> 1
[2, -1] -> -1
[2, 0] -> -1
[2, 1] -> -1
[2, 2] -> 1
[2, 3] -> 1
[2, 4] -> 1
[2, 5] -> -1
[2, 6] -> -1
[2, 7] -> -1
[2, 8] -> -1
[3, -8] -> -1
[3, -7] -> -1
[3, -6] -> -1
[3, -5] -> -1
[3, -4] -> -1
[3, -3] -> 1
[3, -2] -> 1
[3, -1] -> -1
[3, 0] -> 1
[3, 1] -> 1
[3, 2] -> 1
```

```
[3, 3] -> -1
[3, 4] -> -1
[3, 5] -> -1
[3, 6] -> -1
[3, 7] -> -1
[3, 8] -> -1
[4, -8] -> -1
[4, -7] -> -1
[4, -6] -> -1
[4, -5] -> -1
[4, -4] -> -1
[4, -3] -> 1
[4, -2] -> 1
[4, -1] -> 1
[4, 0] -> 1
[4, 1] -> -1
[4, 2] -> -1
[4, 3] -> -1
[4, 4] -> -1
[4, 5] -> -1
[4, 6] -> -1
[4, 7] -> -1
[4, 8] -> -1
[5, -8] -> -1
[5, -7] -> -1
[5, -6] -> -1
[5, -5] -> -1
[5, -4] -> 1
[5, -3] -> 1
[5, -2] -> 1
[5, -1] -> -1
[5, 0] -> -1
[5, 1] -> -1
[5, 2] -> -1
[5, 3] -> -1
[5, 4] -> -1
[5, 5] -> -1
[5, 6] -> -1
[5, 7] -> -1
[5, 8] -> -1
[6, -8] -> -1
[6, -7] -> -1
[6, -6] -> 1
[6, -5] -> 1
[6, -4] -> 1
[6, -3] -> -1
[6, -2] -> -1
[6, -1] -> -1
[6, 0] -> -1
[6, 1] -> -1
```

```
[6, 2] -> -1
[6, 3] -> -1
[6, 4] -> -1
[6, 5] -> -1
[6, 6] -> -1
[6, 7] -> -1
[6, 8] -> -1
[7, -8] -> 1
[7, -7] -> 1
[7, -6] -> 1
[7, -5] -> -1
[7, -4] -> -1
[7, -3] -> -1
[7, -2] -> -1
[7, -1] -> -1
[7, 0] -> -1
[7, 1] -> -1
[7, 2] -> -1
[7, 3] -> -1
[7, 4] -> -1
[7, 5] -> -1
[7, 6] -> -1
[7, 7] -> -1
[7, 8] -> -1
[8, -8] -> 1
[8, -7] -> -1
[8, -6] -> -1
[8, -5] -> -1
[8, -4] -> -1
[8, -3] -> -1
[8, -2] -> -1
[8, -1] -> -1
[8, 0] -> -1
[8, 1] -> -1
[8, 2] -> -1
[8, 3] -> -1
[8, 4] -> -1
[8, 5] -> -1
[8, 6] -> -1
[8, 7] -> -1
[8, 8] -> -1
```

```
Out[174… <matplotlib.collections.PathCollection at 0x140e5f3e0>
```



Can you achieve the same goal with a single hidden layer network? Justify your answer.

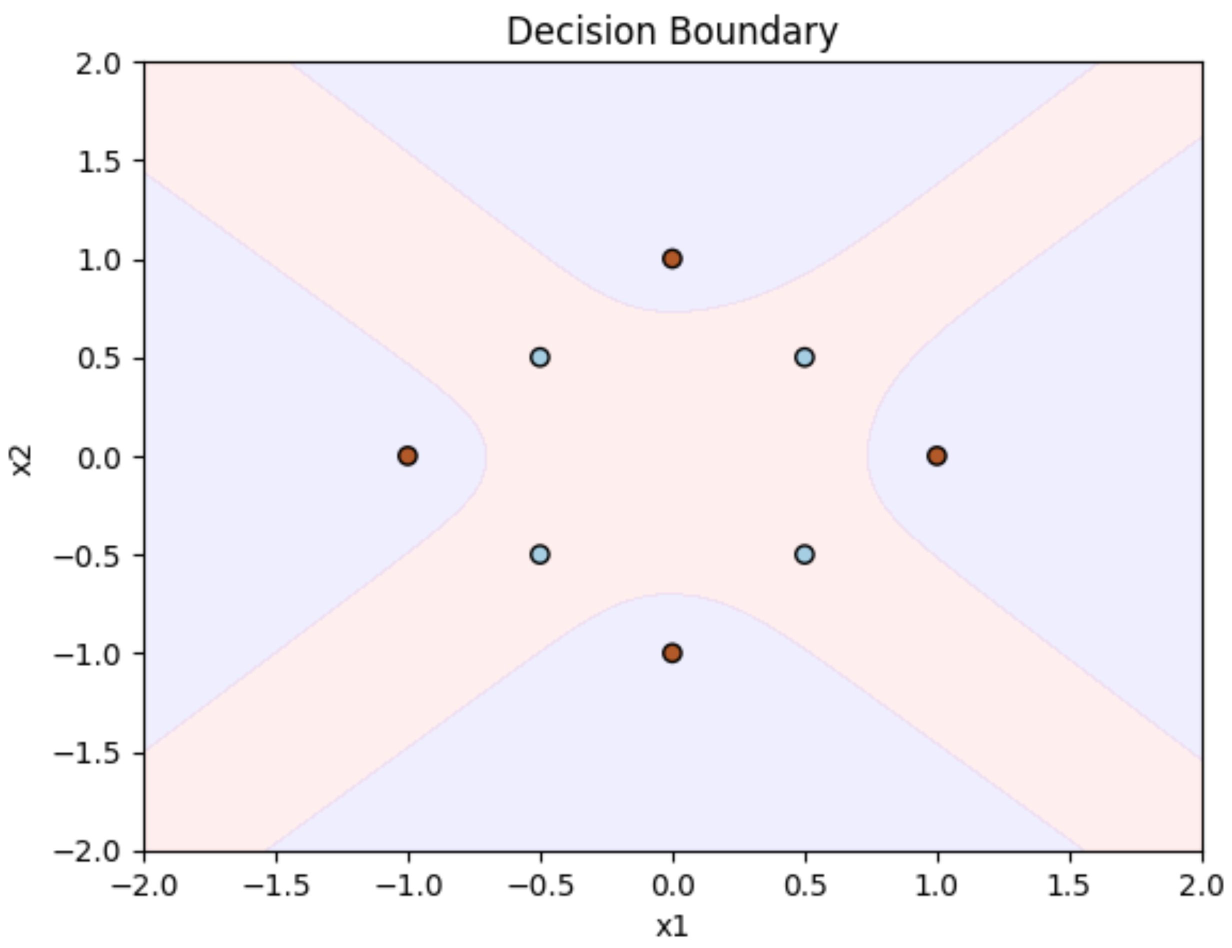
Yes, Single-hidden layer neural networks already possess a universal representation property: by increasing the number of hidden neurons, they can fit (almost) arbitrary functions. You can't get more than this. And particularly not by adding more layers.

In [192…

```
import numpy as np

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Define the training data (STAR problem)
X = np.array([[1, 0],
              [0, 1],
              [-1, 0],
              [0, -1],
              [0.5, 0.5],
              [-0.5, 0.5],
              [0.5, -0.5],
              [-0.5, -0.5]])

# Corresponding labels
y = np.array([[1], [1], [1], [1], [0], [0], [0], [0]])

# Initialize parameters
input_layer_neurons = 2   # Number of input neurons (x1 and x2)
hidden_layer_neurons = 5  # Number of hidden neurons
output_neurons = 1        # Output neuron (binary classification)

# Randomly initialize weights and biases
np.random.seed(1)
W1 = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
b1 = np.random.uniform(size=(1, hidden_layer_neurons))
W2 = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
b2 = np.random.uniform(size=(1, output_neurons))

# Hyperparameters
learning_rate = .1
epochs = 100000  # Number of training iterations

activation_function = 'sigmoid'

# Training the MLP using backpropagation
for epoch in range(epochs):
    # Forward Propagation
    hidden_layer_input = X @ W1 + b1
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = hidden_layer_output @ W2 + b2
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(W2.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_ou

    # Updating Weights and Biases
    W2 += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    b2 += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    W1 += X.T.dot(d_hidden_layer) * learning_rate
    b1 += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

    # Optionally print loss at intervals
    if epoch % 1000 == 0:
        loss = np.mean(np.square(y - predicted_output))
        print(f'Epoch {epoch}, Loss: {loss}')
```

```
# After training, display the final predictions
print("Final predicted output: ")
print(predicted_output)

# Plot the decision boundary
# Define the range for the grid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                     np.linspace(y_min, y_max, 200))

# Flatten the grid to pass through the network
grid = np.c_[xx.ravel(), yy.ravel()]

# Perform forward propagation on the grid data
hidden_layer_input = grid @ W1 + b1
hidden_layer_output = sigmoid(hidden_layer_input)

output_layer_input = hidden_layer_output @ W2 + b2
predicted_output_grid = sigmoid(output_layer_input)

# Reshape the output to match the grid shape
Z = predicted_output_grid.reshape(xx.shape)

# Plot the contour where output is 0.5
plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], alpha=0.2, colors=['#FFAAAA', '#

# Plot the training data
plt.scatter(X[:, 0], X[:, 1], c=y.ravel(), cmap=plt.cm.Paired, edgecolors='k

plt.title('Decision Boundary')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

```
Epoch 0, Loss: 0.3938968948309489
Epoch 1000, Loss: 0.24972614786416378
Epoch 2000, Loss: 0.249221412090127
Epoch 3000, Loss: 0.2463454751881688
Epoch 4000, Loss: 0.21951161194853
Epoch 5000, Loss: 0.17489678746224632
Epoch 6000, Loss: 0.1689347341764805
Epoch 7000, Loss: 0.16779658761925462
Epoch 8000, Loss: 0.16738219722726688
Epoch 9000, Loss: 0.1671781643826819
Epoch 10000, Loss: 0.1670597374214493
Epoch 11000, Loss: 0.1669834983327104
Epoch 12000, Loss: 0.1669307786738204
Epoch 13000, Loss: 0.16689236652251413
Epoch 14000, Loss: 0.16686324622780807
Epoch 15000, Loss: 0.16684047424552823
Epoch 16000, Loss: 0.1668222177363496
Epoch 17000, Loss: 0.16680727968885306
Epoch 18000, Loss: 0.16679484717934953
Epoch 19000, Loss: 0.1667843498641339
Epoch 20000, Loss: 0.16677537643119494
Epoch 21000, Loss: 0.16676762318303878
Epoch 22000, Loss: 0.1667608612542481
Epoch 23000, Loss: 0.16675491506271178
Epoch 24000, Loss: 0.16674964776330228
Epoch 25000, Loss: 0.16674495119592794
Epoch 26000, Loss: 0.16674073879350484
Epoch 27000, Loss: 0.16673694048446744
Epoch 28000, Loss: 0.1667334989671631
Epoch 29000, Loss: 0.16673036694547336
Epoch 30000, Loss: 0.16672750504930395
Epoch 31000, Loss: 0.1667248802505439
Epoch 32000, Loss: 0.16672246464250084
Epoch 33000, Loss: 0.16672023448941528
Epoch 34000, Loss: 0.1667181694790339
Epoch 35000, Loss: 0.16671625212951788
Epoch 36000, Loss: 0.16671446731484485
Epoch 37000, Loss: 0.16671280188203347
Epoch 38000, Loss: 0.1667112443401475
Epoch 39000, Loss: 0.16670978460585872
Epoch 40000, Loss: 0.16670841379391615
Epoch 41000, Loss: 0.1667071240435096
Epoch 42000, Loss: 0.16670590837351068
Epoch 43000, Loss: 0.16670476056107772
Epoch 44000, Loss: 0.1667036750392608
Epoch 45000, Loss: 0.1667026468101203
Epoch 46000, Loss: 0.16670167137055525
Epoch 47000, Loss: 0.16670074464856258
Epoch 48000, Loss: 0.16669986294804878
Epoch 49000, Loss: 0.1666990229006347
Epoch 50000, Loss: 0.1666982214231213
Epoch 51000, Loss: 0.16669745567946193
Epoch 52000, Loss: 0.16669672304619382
Epoch 53000, Loss: 0.1666960210803491
Epoch 54000, Loss: 0.1666953474888625
Epoch 55000, Loss: 0.16669470009843973
Epoch 56000, Loss: 0.1666940768247069
Epoch 57000, Loss: 0.1666934756392227
Epoch 58000, Loss: 0.16669289453254427
Epoch 59000, Loss: 0.16669233147093873
Epoch 60000, Loss: 0.16669178434341198
Epoch 61000, Loss: 0.16669125089432735
Epoch 62000, Loss: 0.16669072863471768
Epoch 63000, Loss: 0.16669021472201384
Epoch 64000, Loss: 0.16668970579254494
Epoch 65000, Loss: 0.16668919772248234
Epoch 66000, Loss: 0.16668868527854727
Epoch 67000, Loss: 0.1666881615955383
Epoch 68000, Loss: 0.16668761737556875
Epoch 69000, Loss: 0.16668703962852127
Epoch 70000, Loss: 0.16668640963389952
Epoch 71000, Loss: 0.16668569953718376
Epoch 72000, Loss: 0.166684866460251
Epoch 73000, Loss: 0.16668384188498206
Epoch 74000, Loss: 0.16668251159901898
Epoch 75000, Loss: 0.16668067563470923
Epoch 76000, Loss: 0.1666779627613582
Epoch 77000, Loss: 0.16667363251073358
Epoch 78000, Loss: 0.166666067619522
Epoch 79000, Loss: 0.16665128836094445
Epoch 80000, Loss: 0.1666177411158373
Epoch 81000, Loss: 0.1665224598233615
Epoch 82000, Loss: 0.16612334001608411
Epoch 83000, Loss: 0.16257661347369604
Epoch 84000, Loss: 0.1261824748395847
Epoch 85000, Loss: 0.01903759846970399
Epoch 86000, Loss: 0.006931446521117218
Epoch 87000, Loss: 0.004314811501028947
Epoch 88000, Loss: 0.0031447326281654623
Epoch 89000, Loss: 0.0024713946474943646
Epoch 90000, Loss: 0.002032524746290394
Epoch 91000, Loss: 0.0017238436004032435
Epoch 92000, Loss: 0.001495082888598071
Epoch 93000, Loss: 0.0013189106253375805
Epoch 94000, Loss: 0.0011791676257595892
Epoch 95000, Loss: 0.0010656857017140565
Epoch 96000, Loss: 0.0009717481606414502
Epoch 97000, Loss: 0.0008927415377778246
Epoch 98000, Loss: 0.0008253933989652753
Epoch 99000, Loss: 0.0007673189911252368
Final predicted output:
[[0.97384829]
 [0.97395129]
 [0.97139641]
 [0.97179539]
 [0.02874353]
 [0.0240432 ]
 [0.02498208]
 [0.02701511]]
```



Is there one possible solution (i.e. a single set of parameters) to exactly separate the two classes of given points?

- No there are infinitely many solutions, and it's not even close as there is not enough data to create a close to a perfect line consistently

Now that you know the location of the decision regions that shatter the data, augment the input data set to make the performance of the MLP more robust, and with faster training. Show experimentally the improvement.

In [203…

```
# 1000 new augmented points by adding Gaussian noise
num_augmented_points = 500
X_augmented = np.vstack([X + np.random.normal(0, 0.1, X.shape) for _ in rang

# augment labels (same as original labels, just replicated)
y_augmented = np.vstack([y for _ in range(num_augmented_points // len(y))])

# combine the original and augmented data
X_aug = np.vstack((X, X_augmented))
y_aug = np.vstack((y, y_augmented))
```

```
# Initialize parameters
input_layer_neurons = 2   # Number of input neurons (x1 and x2)
hidden_layer_neurons = 6  # Number of hidden neurons
output_neurons = 1        # Output neuron (binary classification)

# Randomly initialize weights and biases
np.random.seed(1)
W1 = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
b1 = np.random.uniform(size=(1, hidden_layer_neurons))
W2 = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
b2 = np.random.uniform(size=(1, output_neurons))

# Hyperparameters
learning_rate = .1
epochs = 100000  # Number of training iterations

activation_function = 'sigmoid'

# Training the MLP using backpropagation
for epoch in range(epochs):
    # Forward Propagation
    hidden_layer_input = X_aug @ W1 + b1
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = hidden_layer_output @ W2 + b2
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
    error = y_aug - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(W2.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_ou

    # Updating Weights and Biases
    W2 += hidden_layer_output.T.dot(d_predicted_output) * learning_rate
    b2 += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    W1 += X_aug.T.dot(d_hidden_layer) * learning_rate
    b1 += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

    # Optionally print loss at intervals
    if epoch % 1000 == 0:
        loss = np.mean(np.square(y_aug - predicted_output))
        print(f'Epoch {epoch}, Loss: {loss}')

# After training, display the final predictions
print("Final predicted output: ")
print(predicted_output)
```

```
# Plot the decision boundary
# Define the range for the grid
x_min, x_max = X_aug[:, 0].min() - 1, X_aug[:, 0].max() + 1
y_min, y_max = X_aug[:, 1].min() - 1, X_aug[:, 1].max() + 1

xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                     np.linspace(y_min, y_max, 200))

# Flatten the grid to pass through the network
grid = np.c_[xx.ravel(), yy.ravel()]

# Perform forward propagation on the grid data
hidden_layer_input = grid @ W1 + b1
hidden_layer_output = sigmoid(hidden_layer_input)

output_layer_input = hidden_layer_output @ W2 + b2
predicted_output_grid = sigmoid(output_layer_input)

# Reshape the output to match the grid shape
Z = predicted_output_grid.reshape(xx.shape)

# Plot the contour where output is 0.5
plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], alpha=0.2, colors=['#FFAAAA', '#

# Plot the training data
plt.scatter(X_aug[:, 0], X_aug[:, 1], c=y_aug.ravel(), cmap=plt.cm.Paired, e

plt.title('Decision Boundary Augmented')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

```
Epoch 0, Loss: 0.4435889024570748
Epoch 1000, Loss: 0.08902666903991749
Epoch 2000, Loss: 0.0020646898563543617
Epoch 3000, Loss: 0.0009205408178978298
Epoch 4000, Loss: 0.0005508634245095836
Epoch 5000, Loss: 0.0003875694008402112
Epoch 6000, Loss: 0.00029816937117069383
Epoch 7000, Loss: 0.00024146229222457962
Epoch 8000, Loss: 0.00020157158959006847
Epoch 9000, Loss: 0.00017132810669094213
Epoch 10000, Loss: 0.00014703319165584142
Epoch 11000, Loss: 0.00012667977682540626
Epoch 12000, Loss: 0.00010933110421773961
Epoch 13000, Loss: 9.47552110341071e-05
Epoch 14000, Loss: 8.29890497726262e-05
Epoch 15000, Loss: 7.39166639616636e-05
Epoch 16000, Loss: 6.692407880655511e-05
Epoch 17000, Loss: 6.132083713965073e-05
Epoch 18000, Loss: 5.6683987915079365e-05
Epoch 19000, Loss: 5.276038263314603e-05
Epoch 20000, Loss: 4.938426358185641e-05
Epoch 21000, Loss: 4.644053826471286e-05
Epoch 22000, Loss: 4.384594870899018e-05
Epoch 23000, Loss: 4.153833103324099e-05
Epoch 24000, Loss: 3.947008221808816e-05
Epoch 25000, Loss: 3.760398749514204e-05
Epoch 26000, Loss: 3.591044653980059e-05
Epoch 27000, Loss: 3.4365566964031754e-05
Epoch 28000, Loss: 3.294981683760765e-05
Epoch 29000, Loss: 3.164704969961385e-05
Epoch 30000, Loss: 3.044378488176009e-05
Epoch 31000, Loss: 2.932866708741622e-05
Epoch 32000, Loss: 2.8292054444269185e-05
Epoch 33000, Loss: 2.7325700288207268e-05
Epoch 34000, Loss: 2.6422504405946395e-05
Epoch 35000, Loss: 2.5576316467235928e-05
Epoch 36000, Loss: 2.4781779162295456e-05
Epoch 37000, Loss: 2.403420189003255e-05
Epoch 38000, Loss: 2.3329458197835308e-05
Epoch 39000, Loss: 2.266390186385388e-05
Epoch 40000, Loss: 2.2034297741382567e-05
Epoch 41000, Loss: 2.1437764388888003e-05
Epoch 42000, Loss: 2.087172618151996e-05
Epoch 43000, Loss: 2.0333873105053358e-05
Epoch 44000, Loss: 1.9822126816297807e-05
Epoch 45000, Loss: 1.933461184713037e-05
Epoch 46000, Loss: 1.8869631055438287e-05
Epoch 47000, Loss: 1.8425644602098506e-05
Epoch 48000, Loss: 1.8001251870816734e-05
Epoch 49000, Loss: 1.7595175856274357e-05
Epoch 50000, Loss: 1.7206249632235182e-05
Epoch 51000, Loss: 1.6833404580147284e-05
Epoch 52000, Loss: 1.6475660114099064e-05
Epoch 53000, Loss: 1.6132114682701714e-05
Epoch 54000, Loss: 1.5801937864781652e-05
Epoch 55000, Loss: 1.5484363405413495e-05
Epoch 56000, Loss: 1.5178683063140083e-05
Epoch 57000, Loss: 1.4884241159261155e-05
Epoch 58000, Loss: 1.460042973666262e-05
Epoch 59000, Loss: 1.4326684249435296e-05
Epoch 60000, Loss: 1.4062479716035313e-05
Epoch 61000, Loss: 1.3807327278374201e-05
Epoch 62000, Loss: 1.3560771117320023e-05
Epoch 63000, Loss: 1.332238568192529e-05
Epoch 64000, Loss: 1.3091773195484225e-05
Epoch 65000, Loss: 1.286856140642622e-05
Epoch 66000, Loss: 1.2652401556241575e-05
Epoch 67000, Loss: 1.2442966540208152e-05
Epoch 68000, Loss: 1.2239949239748124e-05
Epoch 69000, Loss: 1.2043061007880952e-05
Epoch 70000, Loss: 1.1852030291499953e-05
Epoch 71000, Loss: 1.1666601376163738e-05
Epoch 72000, Loss: 1.1486533240784819e-05
Epoch 73000, Loss: 1.1311598511071349e-05
Epoch 74000, Loss: 1.1141582501861688e-05
Epoch 75000, Loss: 1.0976282339600045e-05
Epoch 76000, Loss: 1.0815506157186043e-05
Epoch 77000, Loss: 1.0659072354278958e-05
Epoch 78000, Loss: 1.0506808916889465e-05
Epoch 79000, Loss: 1.0358552790750904e-05
Epoch 80000, Loss: 1.021414930354131e-05
Epoch 81000, Loss: 1.0073451631539958e-05
Epoch 82000, Loss: 9.936320306757859e-06
Epoch 83000, Loss: 9.802622760976368e-06
Epoch 84000, Loss: 9.672232903490928e-06
Epoch 85000, Loss: 9.545030729668255e-06
Epoch 86000, Loss: 9.420901957708428e-06
Epoch 87000, Loss: 9.299737691253556e-06
Epoch 88000, Loss: 9.181434105708602e-06
Epoch 89000, Loss: 9.065892156342247e-06
Epoch 90000, Loss: 8.953017306408555e-06
Epoch 91000, Loss: 8.84271927369996e-06
Epoch 92000, Loss: 8.734911794077937e-06
Epoch 93000, Loss: 8.629512400662163e-06
Epoch 94000, Loss: 8.526442217475032e-06
Epoch 95000, Loss: 8.425625766440596e-06
Epoch 96000, Loss: 8.326990786737957e-06
Epoch 97000, Loss: 8.23046806558614e-06
Epoch 98000, Loss: 8.135991279625463e-06
Epoch 99000, Loss: 8.043496846120999e-06
Final predicted output:
[[9.98707050e-01]
 [9.99985287e-01]
 [9.98752493e-01]
 [9.99891522e-01]
 [5.04929384e-06]
 [7.61856567e-05]
 [9.93756726e-04]
 [2.54696489e-08]
 [9.98725148e-01]
 [9.99999948e-01]
 [9.98685182e-01]
 [9.99741177e-01]
 [2.97638265e-06]
 [1.16437432e-04]
 [1.42664213e-03]
 [7.98820788e-06]
 [9.98685065e-01]
```

```
       [9.99972905e-01]
       [9.98803110e-01]
       [9.99837612e-01]
       [8.74985552e-04]
       [8.53962398e-05]
       [6.80016041e-03]
       [4.97471928e-08]
       [9.98728246e-01]
       [9.99987984e-01]
       [9.98610465e-01]
       [9.99848589e-01]
       [8.31467221e-05]
       [3.08188435e-04]
       [9.58558596e-04]
       [2.22884834e-08]
       [9.98612945e-01]
       [9.99989885e-01]
       [9.98803357e-01]
       [9.99900002e-01]
       [4.60437256e-08]
       [1.41329426e-04]
       [1.05123091e-03]
       [2.56747209e-08]
       [9.98705475e-01]
       [9.99981028e-01]
       [9.98252681e-01]
       [9.99866735e-01]
       [2.17514092e-05]
       [2.81064470e-02]
       [1.08276627e-03]
       [2.27895105e-08]
       [9.98730510e-01]
       [9.99991131e-01]
       [9.98052689e-01]
       [9.99893039e-01]
       [4.44192274e-04]
       [7.98905994e-05]
       [1.17119544e-03]
       [8.21337752e-07]
       [9.98720811e-01]
       [9.99983400e-01]
       [9.96976022e-01]
       [9.99867134e-01]
       [3.03654219e-03]
       [1.39767553e-04]
       [9.87572190e-04]
       [1.52769691e-03]
       [9.98705229e-01]
       [9.99980988e-01]
       [9.98633121e-01]
```

```
                [9.99914163e-01]
                [2.81754115e-06]
                [7.81905202e-05]
                [1.28267336e-03]
                [2.99846178e-08]
                [9.98713987e-01]
                [9.99989843e-01]
                [9.98784340e-01]
                [9.99850068e-01]
                [4.70130315e-06]
                [4.64880489e-03]
                [1.09227506e-03]
                [5.79380047e-08]
                [9.98680260e-01]
                [9.99956126e-01]
                [9.98695801e-01]
                [9.99849976e-01]
                [1.99107374e-05]
                [2.88199709e-04]
                [1.30834061e-03]
                [2.88101730e-08]
                [9.97530807e-01]
                [9.99982578e-01]
                [9.98535593e-01]
                [9.99910573e-01]
                [5.92838607e-07]
                [4.28161899e-04]
                [9.83406691e-04]
                [2.19834205e-08]
                [9.98672250e-01]
                [9.99907528e-01]
                [9.97772744e-01]
                [9.99836919e-01]
                [3.56259339e-05]
                [1.35541389e-04]
                [2.95609850e-03]
                [1.58699699e-07]
                [9.98734046e-01]
                [9.99989281e-01]
                [9.98804171e-01]
                [9.96656114e-01]
                [1.45100299e-05]
                [9.20865637e-05]
                [1.28943622e-03]
                [4.56564032e-08]
                [9.98666725e-01]
                [9.99997609e-01]
                [9.98147198e-01]
                [9.91626945e-01]
                [9.48191599e-07]
```

```
        [8.71736911e-05]
        [1.03929974e-03]
        [2.64292593e-06]
        [9.97426536e-01]
        [9.99801800e-01]
        [9.98696473e-01]
        [9.99868974e-01]
        [2.66645737e-06]
        [6.55505603e-04]
        [1.17383468e-03]
        [2.18497447e-07]
        [9.98455802e-01]
        [9.99953131e-01]
        [9.98842583e-01]
        [9.95633323e-01]
        [1.58278761e-07]
        [8.34526935e-05]
        [1.69656756e-03]
        [1.19151516e-07]
        [9.98725223e-01]
        [9.99999571e-01]
        [9.98773446e-01]
        [9.99926782e-01]
        [1.04680097e-04]
        [2.73574487e-04]
        [1.01889707e-03]
        [4.26790443e-10]
        [9.98680898e-01]
        [9.99989457e-01]
        [9.97645507e-01]
        [9.99906440e-01]
        [8.63664317e-06]
        [1.26267257e-04]
        [1.05485919e-03]
        [6.88688173e-08]
        [9.98424938e-01]
        [9.99981055e-01]
        [9.98778345e-01]
        [9.99810470e-01]
        [2.39357862e-04]
        [9.23516987e-05]
        [9.09600140e-04]
        [3.31157675e-08]
        [9.98609731e-01]
        [9.99999488e-01]
        [9.98097834e-01]
        [9.99888753e-01]
        [9.03901117e-05]
        [8.99250683e-05]
        [1.05473387e-03]
```

```
       [9.95634744e-06]
       [9.98533271e-01]
       [9.73152639e-01]
       [9.98355838e-01]
       [9.99880100e-01]
       [8.49073673e-06]
       [7.83887508e-05]
       [1.04774652e-03]
       [3.59276141e-08]
       [9.98718269e-01]
       [9.99815141e-01]
       [9.98764529e-01]
       [9.99904408e-01]
       [8.65961747e-06]
       [9.15327136e-05]
       [2.34486307e-03]
       [5.45051952e-08]
       [9.97911114e-01]
       [9.99995603e-01]
       [9.98468934e-01]
       [9.99151652e-01]
       [4.49792377e-06]
       [1.58665769e-04]
       [9.63153974e-04]
       [1.01116436e-07]
       [9.98699296e-01]
       [9.99916997e-01]
       [9.98797277e-01]
       [9.99853221e-01]
       [2.83529799e-05]
       [1.07266151e-04]
       [1.02991453e-03]
       [5.02295721e-06]
       [9.98741490e-01]
       [9.99900997e-01]
       [9.98688996e-01]
       [9.99820947e-01]
       [1.43091426e-06]
       [7.97344130e-05]
       [1.09817580e-03]
       [1.21056983e-08]
       [9.98688617e-01]
       [9.97323646e-01]
       [9.98416496e-01]
       [9.99674888e-01]
       [1.53886289e-08]
       [4.80508975e-04]
       [1.10501054e-03]
       [1.37895895e-05]
       [9.98725590e-01]
```

```
 [9.99998977e-01]
 [9.98488489e-01]
 [9.99840516e-01]
 [3.82027863e-06]
 [7.98784704e-05]
 [2.51814815e-03]
 [8.68166817e-04]
 [9.98735799e-01]
 [9.99988108e-01]
 [9.98745161e-01]
 [9.99904935e-01]
 [1.09664245e-04]
 [1.93105343e-04]
 [9.17314505e-04]
 [1.25867804e-08]
 [9.98646811e-01]
 [9.99973569e-01]
 [9.98789997e-01]
 [9.99915478e-01]
 [3.22043381e-08]
 [8.38792445e-05]
 [1.08821349e-03]
 [2.63826572e-07]
 [9.98706407e-01]
 [9.99946313e-01]
 [9.98843682e-01]
 [9.99703908e-01]
 [5.83102515e-07]
 [2.84378462e-04]
 [1.06640712e-03]
 [9.21223236e-09]
 [9.98285286e-01]
 [9.99993704e-01]
 [9.98845119e-01]
 [9.99870705e-01]
 [9.81672354e-06]
 [1.38297739e-02]
 [1.03784297e-03]
 [1.01750823e-07]
 [9.98578233e-01]
 [9.99983465e-01]
 [9.98808950e-01]
 [9.99922163e-01]
 [8.14508763e-07]
 [1.26279723e-04]
 [1.26475782e-03]
 [4.04259217e-08]
 [9.97366089e-01]
 [9.99999214e-01]
 [9.98517468e-01]
```

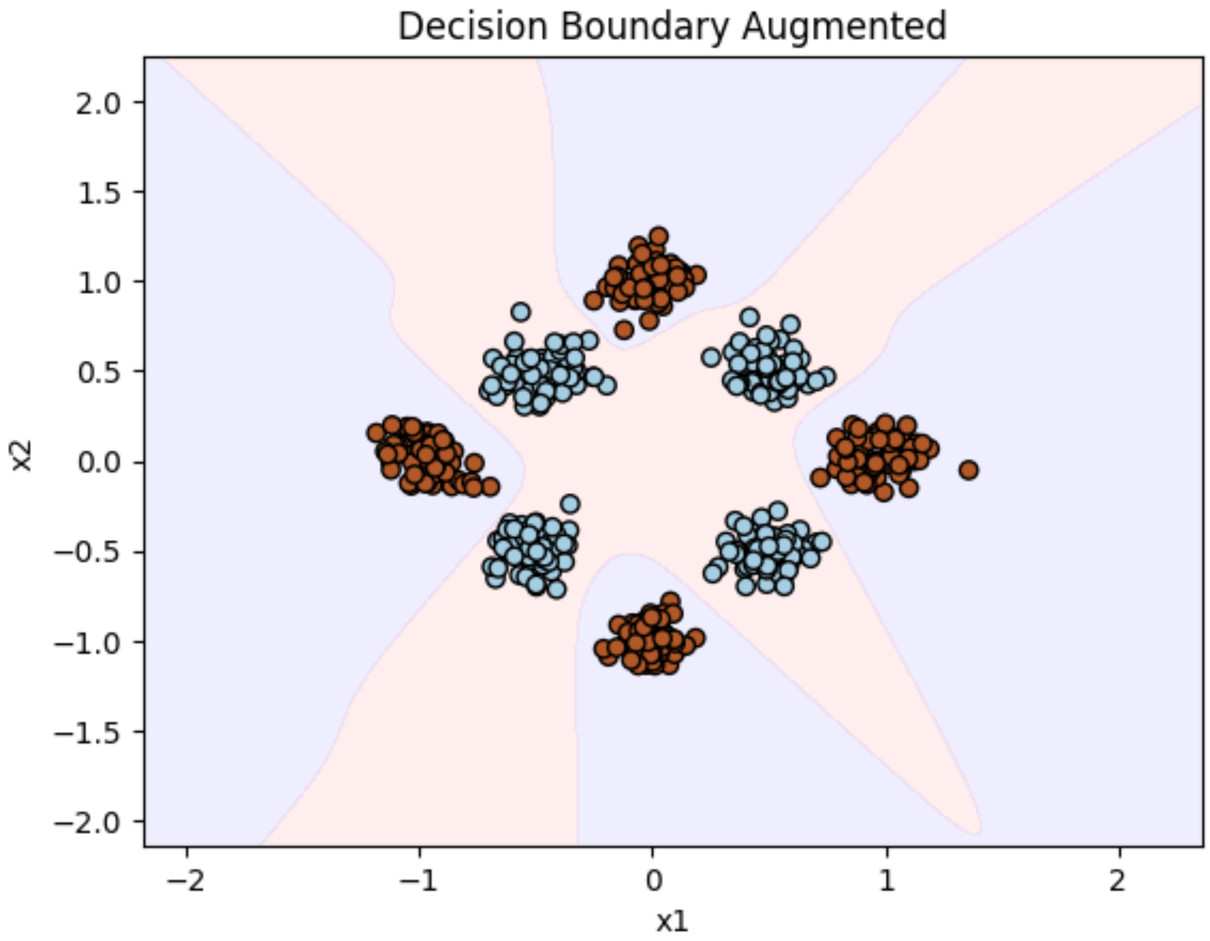
```
                 [9.99324158e-01]
                 [1.75308059e-05]
                 [1.38200406e-04]
                 [7.66800859e-04]
                 [2.19043653e-08]
                 [9.98678073e-01]
                 [9.99926020e-01]
                 [9.98532975e-01]
                 [9.99838779e-01]
                 [2.40572129e-04]
                 [3.91042213e-04]
                 [1.26928806e-03]
                 [1.79677937e-06]
                 [9.98528186e-01]
                 [9.99197604e-01]
                 [9.98447321e-01]
                 [9.99807281e-01]
                 [1.63229501e-03]
                 [1.11244164e-04]
                 [1.00577356e-03]
                 [1.86065939e-07]
                 [9.98734251e-01]
                 [9.99968991e-01]
                 [9.98642782e-01]
                 [9.99904703e-01]
                 [9.34332116e-05]
                 [9.74206814e-05]
                 [1.31514255e-03]
                 [4.05716255e-08]
                 [9.98713251e-01]
                 [9.99753214e-01]
                 [9.98485036e-01]
                 [9.99821317e-01]
                 [2.29017720e-05]
                 [1.54075536e-04]
                 [1.14591360e-03]
                 [1.69823317e-07]
                 [9.98735922e-01]
                 [9.99999420e-01]
                 [9.98477098e-01]
                 [9.99876131e-01]
                 [1.47358352e-07]
                 [2.14024127e-04]
                 [1.11315063e-03]
                 [1.82118790e-04]
                 [9.98580489e-01]
                 [9.99690504e-01]
                 [9.98786293e-01]
                 [9.99801762e-01]
                 [1.67124827e-02]
```

```
            [1.00022324e-04]
            [9.62306490e-04]
            [3.57594979e-08]
            [9.98730238e-01]
            [9.99991073e-01]
            [9.98617684e-01]
            [9.99882101e-01]
            [1.53339006e-05]
            [2.67329103e-04]
            [1.04904568e-03]
            [2.53765799e-07]
            [9.98128205e-01]
            [9.99976208e-01]
            [9.97514713e-01]
            [9.99696121e-01]
            [2.03415406e-06]
            [7.52621933e-05]
            [6.32165300e-03]
            [1.26600691e-06]
            [9.98454797e-01]
            [9.99994172e-01]
            [9.98396270e-01]
            [9.99917571e-01]
            [3.47802273e-05]
            [4.29163094e-04]
            [1.10274890e-03]
            [1.27232727e-06]
            [9.98667673e-01]
            [9.79238051e-01]
            [9.98346713e-01]
            [9.99474432e-01]
            [1.12052978e-07]
            [8.87058519e-05]
            [1.00852539e-03]
            [2.55147365e-03]
            [9.98729125e-01]
            [9.99979203e-01]
            [9.95372156e-01]
            [9.99900267e-01]
            [1.27452965e-09]
            [2.87377512e-03]
            [1.01367524e-03]
            [6.29830708e-06]
            [9.98730459e-01]
            [9.99869643e-01]
            [9.98647249e-01]
            [9.99910700e-01]
            [3.91038308e-06]
            [9.57867444e-05]
            [1.89019291e-03]
```

```
                 [2.23822638e-07]
                 [9.98546392e-01]
                 [9.99946001e-01]
                 [9.98371663e-01]
                 [9.99859958e-01]
                 [4.32999481e-10]
                 [7.93009156e-05]
                 [1.01062563e-03]
                 [7.41162766e-06]
                 [9.98500496e-01]
                 [9.99980692e-01]
                 [9.98820592e-01]
                 [9.99832376e-01]
                 [2.79021749e-06]
                 [2.00896056e-03]
                 [9.94520575e-04]
                 [2.26711508e-07]
                 [9.98040276e-01]
                 [9.99987440e-01]
                 [9.98413531e-01]
                 [9.99909663e-01]
                 [1.88929866e-05]
                 [3.70675007e-03]
                 [9.61854388e-04]
                 [7.13433474e-08]
                 [9.98551248e-01]
                 [9.99876365e-01]
                 [9.98804182e-01]
                 [9.99872689e-01]
                 [9.06600170e-03]
                 [1.36176791e-04]
                 [1.12574161e-03]
                 [9.82186703e-04]
                 [9.98555286e-01]
                 [9.99993174e-01]
                 [9.98453789e-01]
                 [9.99889090e-01]
                 [1.50891815e-04]
                 [2.80131830e-04]
                 [1.01231073e-03]
                 [3.27303293e-08]
                 [9.98683371e-01]
                 [9.99999611e-01]
                 [9.98661743e-01]
                 [9.99796533e-01]
                 [5.23020736e-05]
                 [1.42276130e-04]
                 [1.06474107e-03]
                 [7.59828610e-08]
                 [9.98336862e-01]
```

```
               [9.99989940e-01]
               [9.98662566e-01]
               [9.99072715e-01]
               [1.33047319e-05]
               [3.32890817e-04]
               [9.36452044e-04]
               [6.99807439e-08]
               [9.98688184e-01]
               [9.99801093e-01]
               [9.98686219e-01]
               [9.99838432e-01]
               [4.30390709e-06]
               [5.25504419e-03]
               [1.80192694e-02]
               [1.74036553e-07]
               [9.98680839e-01]
               [9.99974673e-01]
               [9.98831849e-01]
               [9.99753525e-01]
               [5.72828312e-06]
               [1.83362279e-03]
               [1.00192818e-03]
               [7.30242205e-06]
               [9.98727445e-01]
               [9.99998529e-01]
               [9.98234746e-01]
               [9.99839270e-01]
               [1.15160731e-05]
               [1.15343653e-04]
               [1.00068469e-03]
               [2.96144954e-07]
               [9.98262618e-01]
               [9.99991939e-01]
               [9.98067874e-01]
               [9.99802227e-01]
               [6.74959771e-10]
               [8.04891059e-04]
               [1.15048321e-03]
               [3.19201045e-08]
               [9.78057216e-01]
               [9.99988369e-01]
               [9.97686806e-01]
               [9.98835756e-01]
               [1.48540548e-05]
               [7.91612017e-05]
               [1.11391392e-03]
               [3.05163844e-07]
               [9.98684988e-01]
               [9.99989309e-01]
               [9.98852006e-01]
```

```
 [9.99709178e-01]
 [6.19283380e-04]
 [7.89535207e-05]
 [1.22152736e-03]
 [5.49661180e-05]
 [9.98625105e-01]
 [9.99996442e-01]
 [9.98833505e-01]
 [9.99838014e-01]
 [3.25870480e-05]
 [7.90230535e-05]
 [1.07682498e-03]
 [6.80707083e-08]
 [9.98329282e-01]
 [9.99943619e-01]
 [9.98644914e-01]
 [9.99878865e-01]
 [1.43233441e-08]
 [9.48210426e-04]
 [9.85621127e-04]
 [9.13573478e-08]
 [9.98056242e-01]
 [9.99937782e-01]
 [9.97956288e-01]
 [9.99714615e-01]
 [3.05530773e-10]
 [7.71171492e-05]
 [8.62202865e-04]
 [2.64436832e-08]
 [9.98209382e-01]
 [9.99972196e-01]
 [9.98370453e-01]
 [9.99763126e-01]
 [4.27404156e-10]
 [8.07891909e-05]
 [1.27682526e-03]
 [1.17288414e-07]]
```



Decision boundary looks similar, but in this case it's more precise as now the data isn't in a perfect star pattern. So the model fit it more closely. To get the X shape like the prior graph, I added another node in the hidden layer to provide it with more info to train properly.